PISO-730/730A

User's Manual

Warranty

All products manufactured by ICP DAS are warranted against defective materials for a period of one year from the date of delivery to the original purchaser.

Warning

ICP DAS assumes no liability for damages consequent to the use of this product. ICP DAS reserves the right to change this manual at any time without notice. The information furnished by ICP DAS is believed to be accurate and reliable. However, no responsibility is assumed by ICP DAS for its use, not for any infringements of patents or other rights of third parties resulting from its use.

Copyright

Trademark

The names used for identification only may be registered trademarks of their respective companies.

Tables of Contents

1. Introduction

The PISO-730 provides 32-channel isolated digital I/O (16×DI and 16×DO) and 32-channel TTL-level digital I/O (16×DI and 16×DO). Each digital output offers a NPN transistor and an integral suppression diode for inductive loads. The PISO-730A provides 32-channel isolated digital I/O (16×DI and 16×DO) and 32-channel TTL-level digital I/O (16×DI and 16×DO). Each digital output offers a PNP transistor and an integral suppression diode for inductive loads. Both boards interface to field logic signals, eliminate any ground-loop problems and isolate the host PC from damaging voltages. The isolated I/O channels provide up to 3750Vdc of protection.

The PISO-730/730A has one 37-pin D-type connector and two 20-pin flat-cable connectors. The flat-cable can be connected to an ADP-20/PCI adapter. The adapter can be fixed on the chassis, installs in a 5V PCI slot and can support true Plug and Play".

1.1 Features

- PCI Bus
- 32 isolated DIO channels (16×DI and 16×DO)
- 32 TTL-level DIO channels (16×DI and 16×DO)
- DC/DC converter build-in
- One DB-37 D-type connector for isolated input and output
- Two separate 20-pin connectors for non-isolated input and output
- Up to 3750Vdc isolated voltage
- Interrupt source: 2 channels
- Connects directly to DB-24PR, 24POR, DB-24C, DB-16P, DB-16R
- SMD, short card, power saving
- Automatically detected by Windows 95/98/NT
- No base address or IRQ switches to set

1.2 Specifications

Isolated DIO channels

- **Optical isolated input channel**
  Channel Number: 16 digital inputs
  Type: Isolated current input
  Isolation voltage: 3750Vdc
  Input voltage: 5-30Vdc
  Input impedance: 1.2KΩ/1W
  Response time: 10KHz (Max.)
- **Optical isolated output channel**
  Channel Number: 16 digital outputs
  Output voltage: open-collector 10-30Vdc
  Isolation voltage: 3750Vdc
  Current Sinking (PISO-730): 100mA (Max.)
  Current Sourcing (PISO-730A): 100mA(Max.)
  Response time: 10KHz (Max.)

TTL-level DIO channels

- **TTL-level input channel**
  Channel Number: 16 digital inputs
  Input voltage level: (TTL compatible)
  V_{IL}: 0.8V (Max.)
  V_{IH}: 2.4V (Min.)
- **TTL-level output channel**
  Channel Number: 16 digital outputs
  Driver capacity: (TTL compatible)
  I_{OL}: 8 mA (sink)
  I_{OH}: 0.4 mA (source)

General specifications

- Operation Temp: 0-50°C
- Storage Temp: -20°C to 70°C
- Humidity: 0-90%, non-condensing
- Dimensions: 180mm×105mm
- Power consumption of PISO-730: 640mA
- Power consumption of PISO-730A: 600mA

1.3 Order Description

- PISO-730
  16-channel isolated digital input & 16 channel isolated digital output (current sinking) board
- PISO-730A:
  16-channel isolated digital input & 16 channel isolated digital output (current sourcing) board

1.3.1 Options

- DB-24PR, DB-24PR: 24-channel power relay board
- DB-24POR: 24-channel PhotoMos output board
- DB-24C: 24-channel open-collector output board
- DB-16P: 16-channel isolated D/I board
- DB-16R: 16-channel relay board
- ADP-20/PCI: Extender, 20-pin header to 20-pin header for PCI bus I/O boards
- DN-37: I/O connector block with DIN-Rail mounting and 37-pin D-type connector
- DB-37: 37-pin D-type connector pin-to-pin screw terminal for any 37-pin D-type connector I/O board

1.4 PCI Data Acquisition Family

We provide a family of PCI-BUS data acquisition cards. These cards can be divided into three groups as follows:

1. **PCI-series: first generation, isolated or non-isolated cards**
   PCI-1002/1202/1800/1802/1602: multi-function family, non-isolated
   PCI-P16R16/P16C16/P16POR16/P8R8: D/I/O family, isolated
   PCI-TMC12: timer/counter card, non-isolated
2. **PIO-series: cost-effective generation, non-isolated cards**
   PIO-823/821: multi-function family
   PIO-D144/D96/D64/D56/D48/D24: D/I/O family
   PIO-DA16/DA8/DA4: D/A family
3. **PISO-series: cost-effective generation, isolated cards**
   PISO-813: A/D card
   PISO-P32C32/P32A32/P64/C64/A64: D/I/O family
   PISO-P8R8/P8SSR8AC/P8SSR8DC: D/I/O family
   PISO-730/730A: D/I/O card
   PISO-DA2: D/A card

1.5 Product Check list

In addition to this manual, the package includes the following items:

- One PISO-730/PISO-730A card
- One company floppy diskette or CD
- One release note

  Please read the release note first. All of the following important information will be given in the release note:

1. The location of the software driver & utility
2. How to install software & utility
3. The location of the diagnostic program
4. FAQ

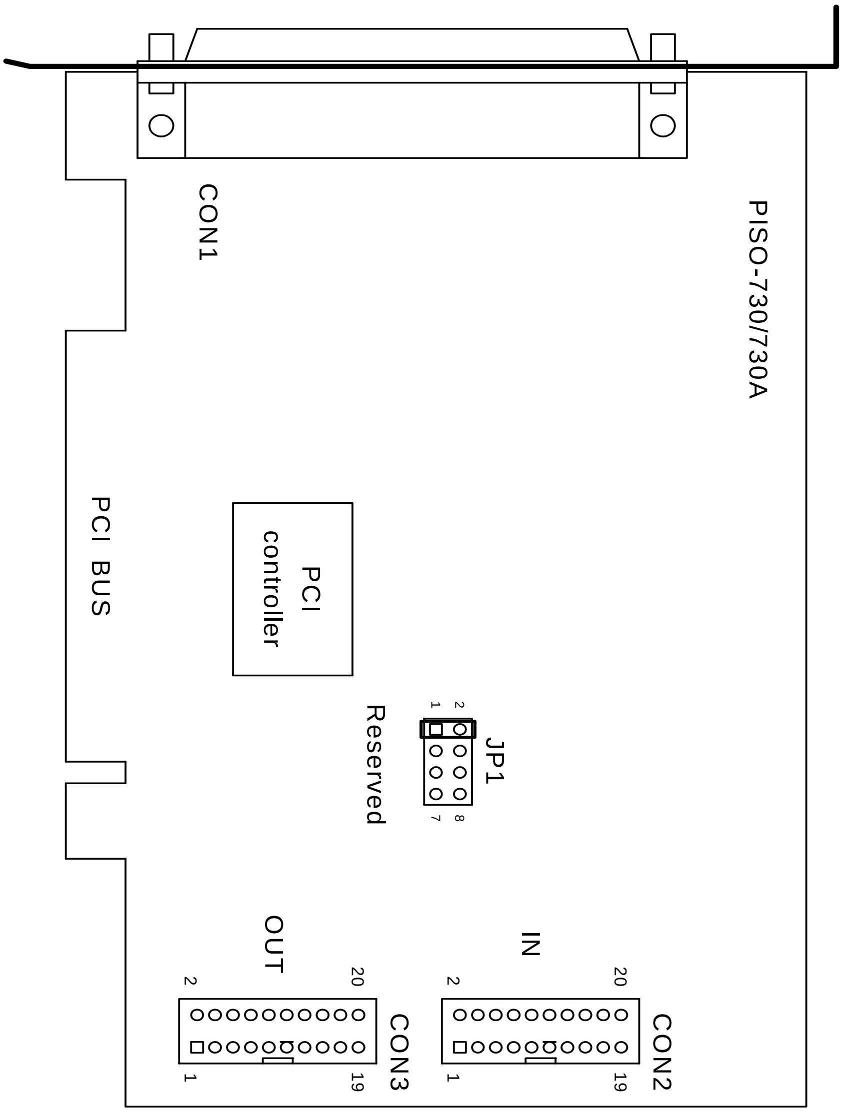
Attention!

If any of these items are missing or damaged, contact the dealer from whom you purchased the product. Save all shipping materials and the carton in case you want to ship or store the product in the future.

2. Hardware configuration

2.1 Board Layout

2.1.1



CON1: 16-channel isolated D/I and 16-channel isolated D/O

CON2: 16-channel TTL-level (non-isolated) D/I

CON3: 16-channel TTL-level (non-isolated) D/O

JP1: Reserved

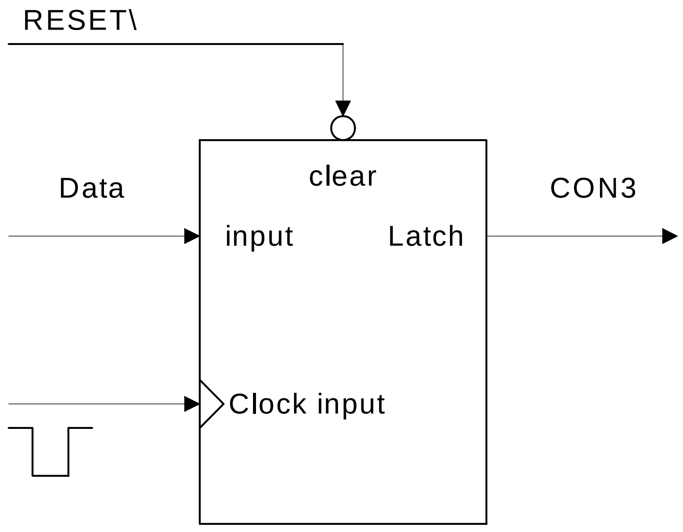
2.2 I/O Operation

2.2.1 Non-isolated DO Port Architecture (CON3)

When the PC is powered-up, all operations of non-isolated DO states are cleared to low state. The RESET\ signal is used to clear non-isolated DO states. Refer to Sec. 3.3.1 for more information about the RESET\ signal.

- The RESET\ is in Low-state → all non-isolated DO states are clear to low state

Here's block diagram of the non-isolated DO:

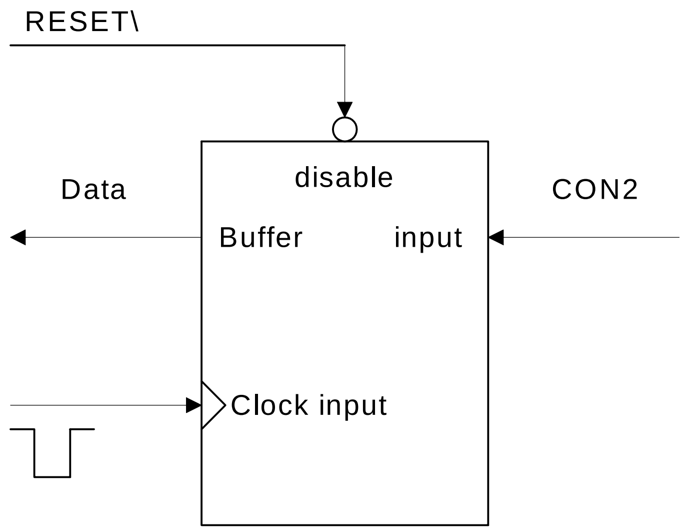


D/O buffer CKT

2.2.2 Non-isolated DI Port Architecture (CON2)

When the PC is powered-up, non-isolated DI port operations are disabled. The enable/disable for non-isolated DI ports is controlled by the RESET\ signal. Refer to Sec. 3.3.1 for more information about the RESET\ signal.

- The RESET\ is in Low-state → all non-isolated DI operation are disabled
- The RESET\ is in High-state → all non-isolated DI operation are enabled



D/I buffer CKT

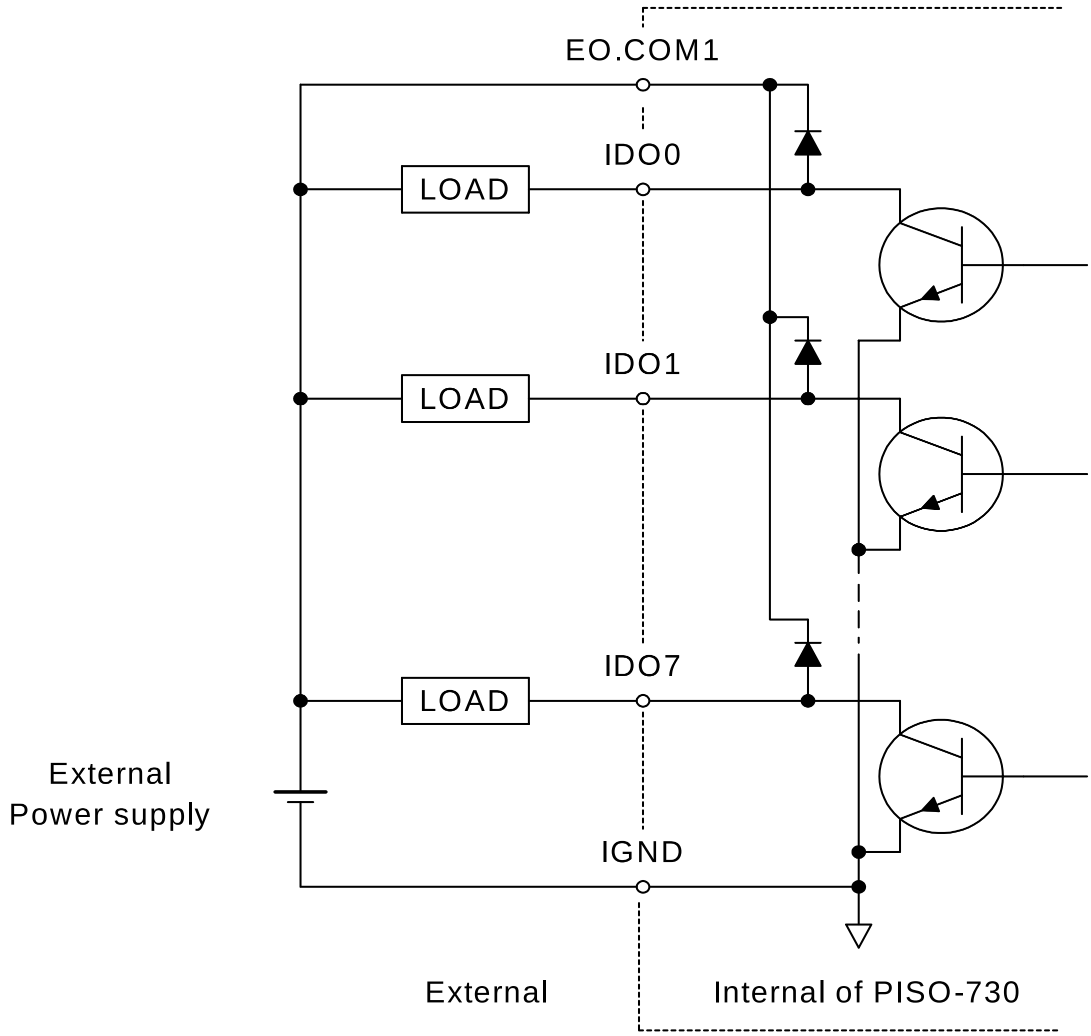
2.2.3 Isolated DO Port Architecture (CON1)

When the PC is powered-up, all operations of isolated DO states are cleared to low state. The RESET\ signal is used to clear isolated DO states. Refer to Sec. 3.3.1 for more information about RESET\ signal.

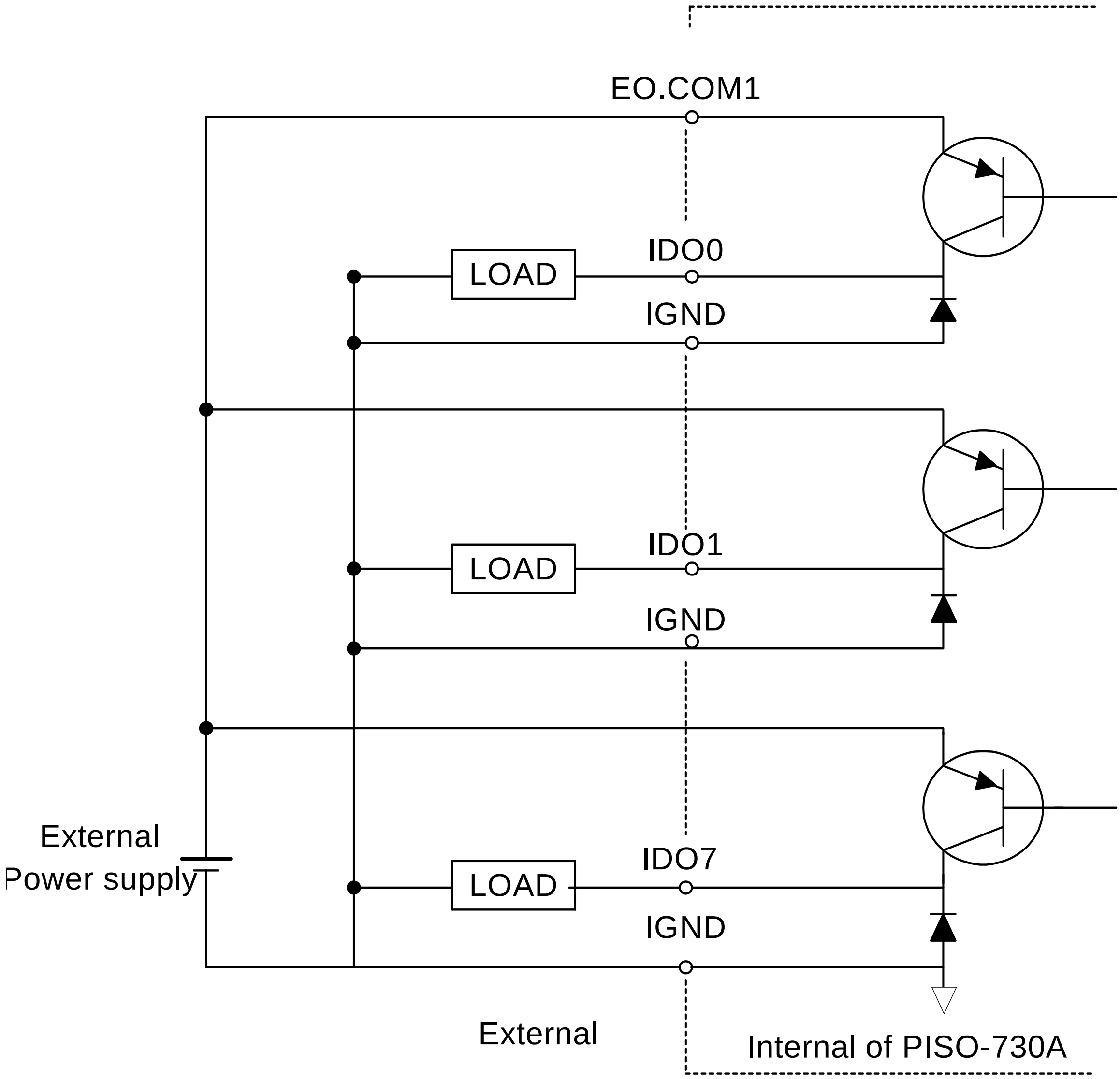
- The RESET\ is in Low-state → all isolated DO states are clear to low state

Each eight open-collector output channel shares EO.COM (IDO0~IDO7 use EO.COM1 and IDO8~IDO15 use EO.COM2)

The block diagram of isolated DO (current sinking) is as follows:

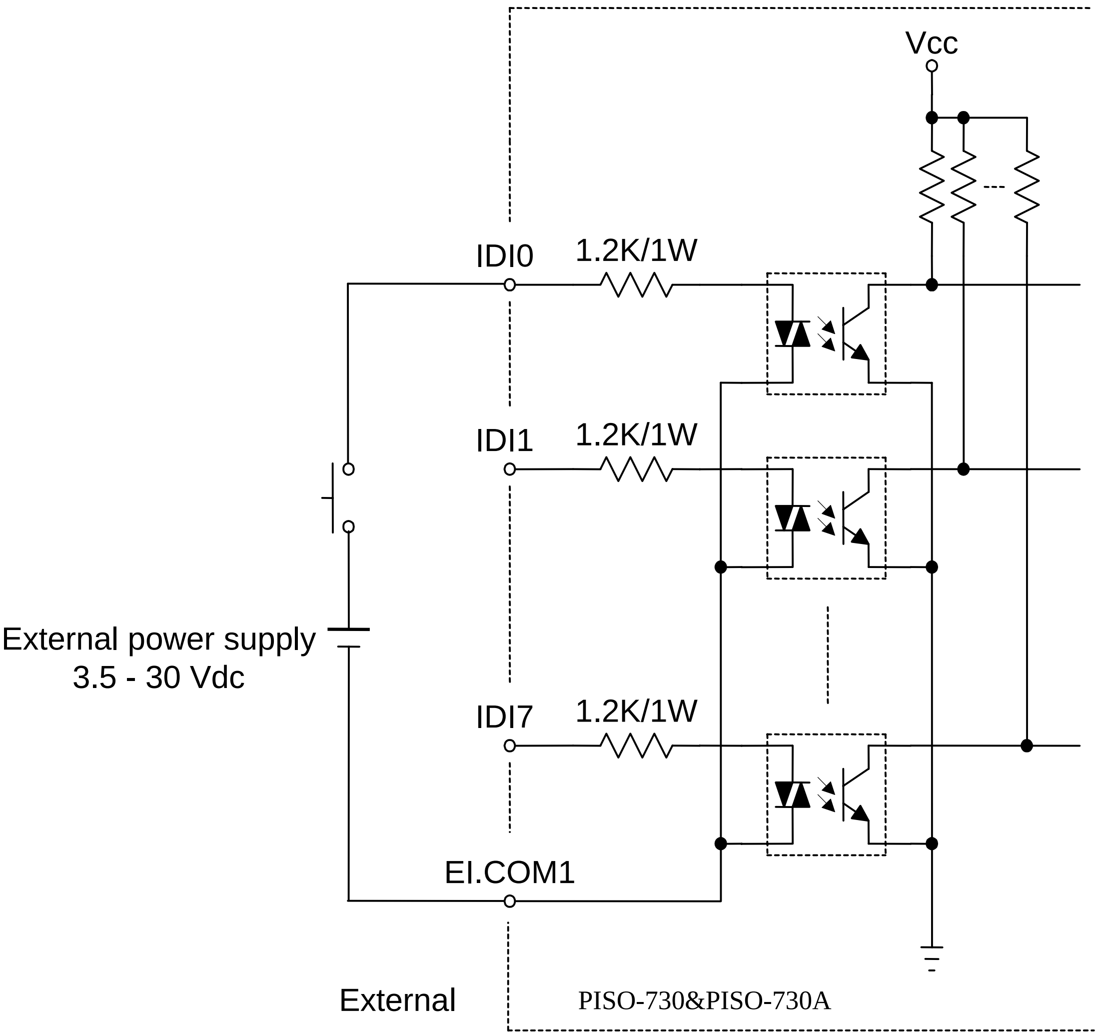


The block diagram of isolated DO (current sourcing) is as follows:



2.2.4 Isolated DI Port Architecture (CON1)

The PISO-730/730A provides 16-channel isolated digital input. Each of the isolated digital input can accept voltages from 3.5-30Vdc. Each eight input channels share one external common end point. (IDI0~IDI7 use EI.COM1 and IDI8~IDI15 use EI.COM2)



2.3 Interrupt Operation

There are two interrupt sources in PISO-730/730A. These two signals are named as INT_CHAN_0 and INT_CHAN_1. Their signal sources are given as follows:

INT_CHAN_0: DI0
INT_CHAN_1: DI1

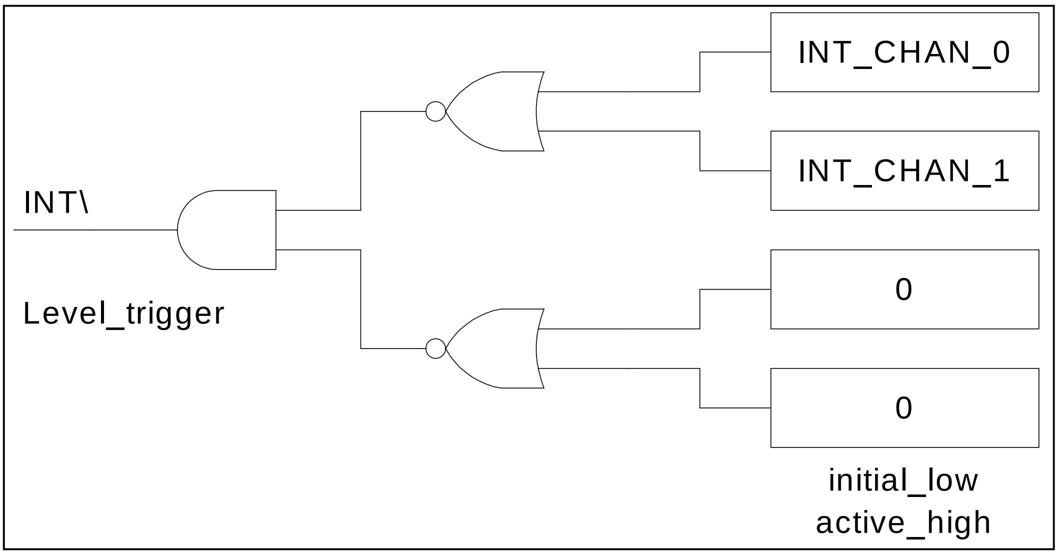
If only one interrupt signal source is used, the interrupt service routine does not have to identify the interrupt source. Refer to DEMO3.C and DEMO4.C for more information.

If there is more than one interrupt source, the interrupt service routine will identify the active signals as follows: (refer to DEMO5.C)

1. Reads the new status of all interrupt signal sources (refer to Sec 3.3.5)
2. Compares the new status with the old status to identify the active signals
3. If INT_CHAN_0 is active, services it
4. If INT_CHAN_1 is active, services it
5. Updates interrupt status

Note: if the interrupt signal is too short, the new status may be as same as old status. In that condition, the interrupt service routine can not identify which interrupt source is active. So the interrupt signal must be hold_active long enough until the interrupt service routine is executed. This hold_time is different for different O.S. The hold_time can be as short as a micro-second or as long as a second. In general, 20ms is enough for any O. S.

2.3.1 Interrupt Block Diagram of PISO-730/730A



The interrupt output signal of PISO-730/730A, INT\ is a **level-trigger, Active_Low signal**. If the INT\ generates a low-pulse, the PISO-730 will interrupt the PC once a time. If the INT\ is fixed in low level, the PISO-730 will interrupt the PC continuously. So the INT_CHAN_0/1 must be controlled by **pulse_type** signals. **They must be fixed in low-level state normally and generate a high_pulse to interrupt the PC.**

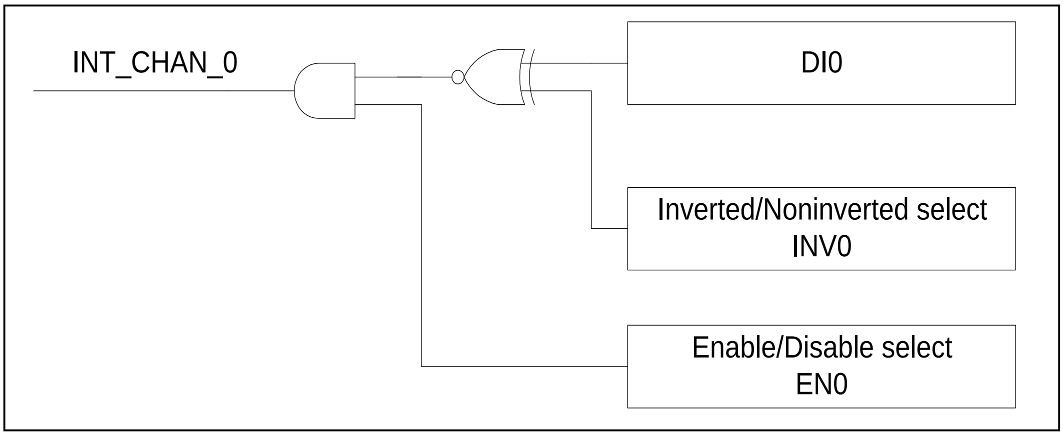
The priority of INT_CHAN_0/1 is the same. If these two signals are active at the same time, then INT\ will be activated only once. So the interrupt service routine has to read the status of all interrupt channels for a multi-channel interrupt. Refer to Sec. 2.6.7 for more information.

DEMO5.C → for multi-channel interrupt source

If only one interrupt source is used, the interrupt service routine doesn't have to read the status of interrupt source. The demo programs DEMO3.C and DEMO4.C are designed for single-channel interrupt demo as follows:

DEMO3.C & DEMO4.C → for INT_CHAN_0 only

2.3.2 INT_CHAN_0



The INT_CHAN_0 must be fixed in a normal, low-level state and generate a high_pulse to interrupt the PC.

The EN0 can be used to enable/disable the INT_CHAN_0 as follows: (refer to Sec. 3.3.4)

EN0=0→INT_CHAN_0=disable

EN0=1→INT_CHAN_0=enable

The INV0 can be used to invert/non-invert the DI0 as follows: (Refer to Sec. 3.3.6)

INV0=0→INT_CHAN_0=invert state of DI0

INV0=1→INT_CHAN_0=non-invert state of DI0

Refer to the following demo program for more information:

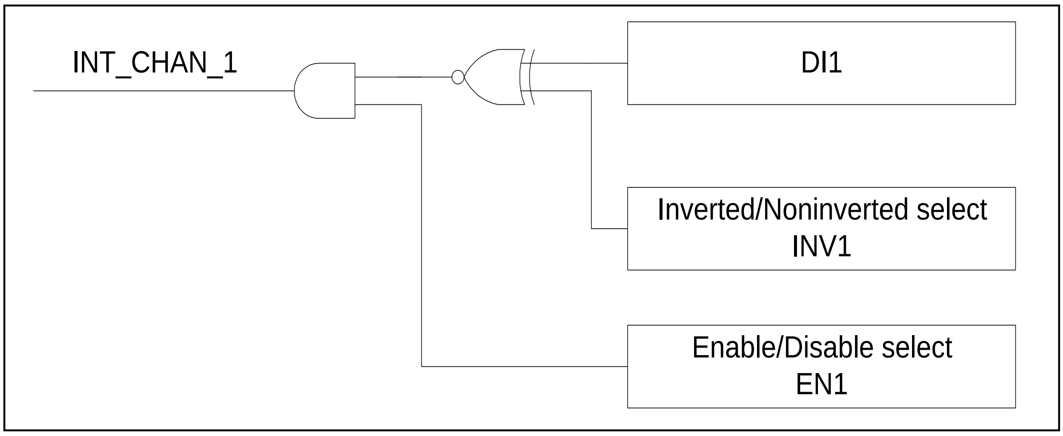
DEMO3.C → for INT_CHAN_0 (initial high)

DEMO4.C → for INT_CHAN_0 (initial low)

DEMO5.C → for multi-channel interrupt source

NOTE: Refer to Sec. 2.3.4 & Sec. 2.3.5 for active high-pulse generation.

2.3.3 INT_CHAN_1



The INT_CHAN_1 must be fixed in a normal low-level state and generated a high_pulse to interrupt the PC.

The EN1 can be used to enable/disable the INT_CHAN_1 as follows: (refer to Sec. 3.3.4)

EN1=0→INT_CHAN_1=disable

EN1=1→INT_CHAN_1=enable

The INV1 can be used to invert/non-invert the DI1 as follows: (Refer to Sec. 3.3.6)

INV1=0→INT_CHAN_1=invert state of DI1

INV1=1→INT_CHAN_1=non-invert state of DI1

Refer to demo program for more information as follows:

DEMO3.C → for INT_CHAN_0 (initial high)

DEMO4.C → for INT_CHAN_0 (initial low)

DEMO5.C → for multi-channel interrupt source

<u>**NOTE: Refer to Sec. 2.3.4 & Sec. 2.3.5 for active high-pulse generation.**</u>

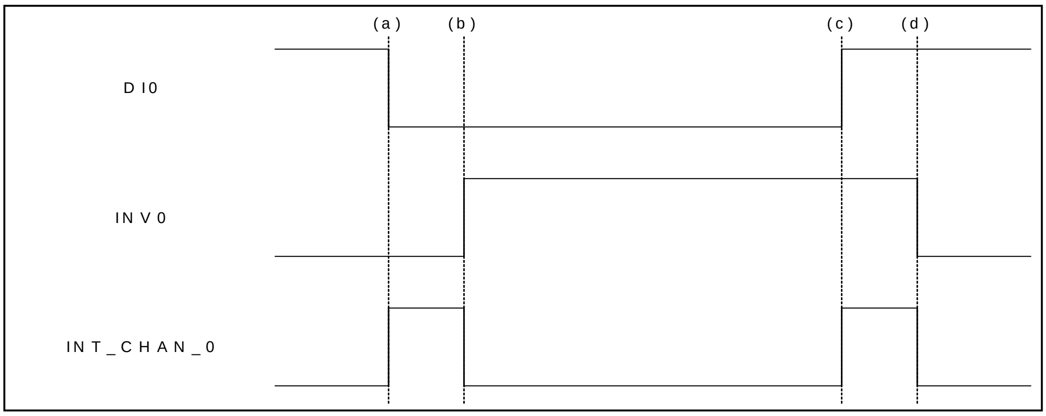
2.3.4 Initial_high, active_low Interrupt source

If the DI0 is an initial_high, active_low signal, the interrupt service routine should use INV0 to invert/non-invert the DI0 for high_pulse generation as follows: (Refer to DEMO3.C and the DI1 is similarly)

Initial set:

```
now_int_state=1;                /* initial state for DI0      */
outportb(wBase+0x2a,0);         /* select the inverted DI0  */
```

```
void interrupt irq_service()
{
if (now_int_state==1)       /* now DI0 is changed to LOW          */(a)
  {                         /* --> INT_CHAN_0=!DI0=HIGH now       */
  COUNT_L++;                /* find a LOW_pulse (DI0)             */
  If((inport(wBase+7)&1)==0)/* the DI0 is still fixed in LOW      */
    {                       /* → needs to generate a high_pulse   */
    outportb(wBase+0x2a,1);/* INV0 select the non-inverted input */(b)
                            /* INT_CHAN_0=DI0=LOW -->             */
                            /* INT_CHAN_0 generate a high_pulse   */
    now_int_state=0;        /* now DI0=LOW                        */
    }
  else now_int_state=1;     /* now DI0=HIGH                       */
                            /* doesn    have to generate high_pulse  */
  }
else                        /* now DI0 is changed to HIGH         */(c)
  {                         /* --> INT_CHAN_0=DI0=HIGH now        */
  COUNT_H++;                /* find a HIGH_pulse (DI0)            */
  If((inport(wBase+7)&1)==1)/* the DI0 is still fixed in HIGH     */
    {                       /* needs to generate a high_pulse     */
    outportb(wBase+0x2a,0);/* INV0 select the inverted input     */(d)
                            /* INT_CHAN_0=!DI0=LOW -->            */
                            /* INT_CHAN_0 generate a high_pulse   */
    now_int_state=1;        /* now DI0=HIGH                       */
    }
  else now_int_state=0;     /* now DI0=LOW                        */
                            /* doesn    have to generate high_pulse  */
  }
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```



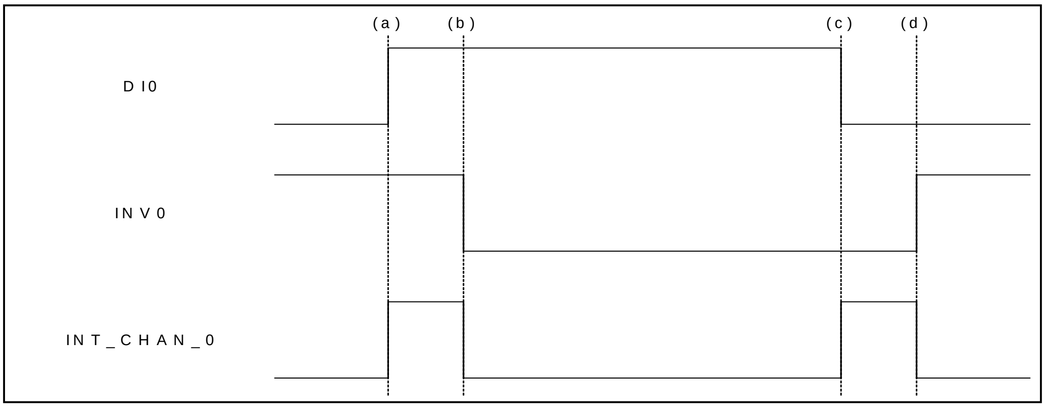
2.3.5 Initial_low, active_high Interrupt source

If the DI0 is an initial_low, active_high signal, the interrupt service routine should use INV0 to invert/non-invert the DI0 for high_pulse generation as follows: (Refer to DEMO4.C and the DI1 is similarly)

Initial set:

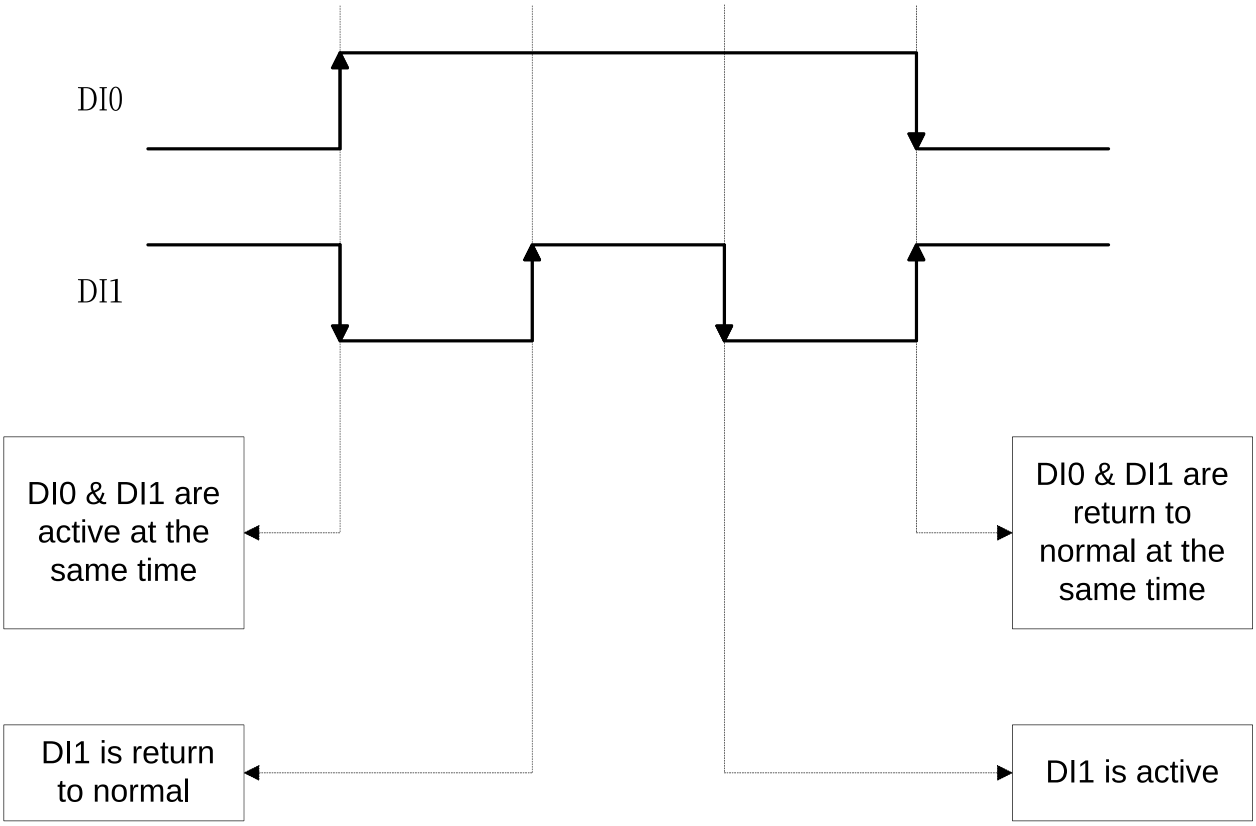
```
now_int_state=0;                /* initial state for DI0         */
outportb(wBase+0x2a,1);         /* select the non-inverted DI0 */
```

```
void interrupt irq_service()
{
if (now_int_state==1)      /* now DI0 is changed to LOW         */(c)
  {                  /* --> INT_CHAN_0=!DI0=HIGH now      */
  COUNT_L++;              /* find a LOW_pulse (DI0)           */
  If((inport(wBase+7)&1)==0)/* the DI0 is still fixed in LOW    */
   {                 /* → needs to generate a high_pulse   */
   outportb(wBase+0x2a,1);/* INV0 select the non-inverted input */(d)
                     /* INT_CHAN_0=DI0=LOW -->             */
                     /* INT_CHAN_0 generate a high_pulse   */
   now_int_state=0;      /* now DI0=LOW                       */
   }
  else now_int_state=1;   /* now DI0=HIGH                      */
                     /* doesn    have to generate high_pulse  */
  }
else                    /* now DI0 is changed to HIGH        */(a)
  {                  /* --> INT_CHAN_0=DI0=HIGH now        */
  COUNT_H++;              /* find a High_pulse (DI0)          */
  If((inport(wBase+7)&1)==1)/* the DI0 is still fixed in HIGH   */
   {                 /* needs to generate a high_pulse      */
   outportb(wBase+0x2a,0);/* INV0 select the inverted input    */(b)
                     /* INT_CHAN_0=!DI0=LOW -->            */
                     /* INT_CHAN_0 generate a high_pulse   */
   now_int_state=1;      /* now DI0=HIGH                      */
   }
  else now_int_state=0;   /* now DI0=LOW                       */
                     /* doesn    have to generate high_pulse  */
  }
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```



2.3.6 Multiple Interrupt Source

Assume: DI0 is initial Low, active High
DI1 is initial High, active Low
as follows:



Refer to DEMO5.C for source program. **These three falling-edge & rising-edge scenarios can be detected by DEMO5.C.**

Note: When the interrupt is active, the user program has to identify the active signals. More than one signal may be simultaneously. The interrupt service routine has to service all active signals at the same time.

Initial setting:

```
now_int_state=0x2;              /* Initial state: DI0 at low level, DI1 at high level   */
invert=0x1;                     /* non-invert DI0 & invert DI1                           */
outportb(wBase+0x2a,invert);
```

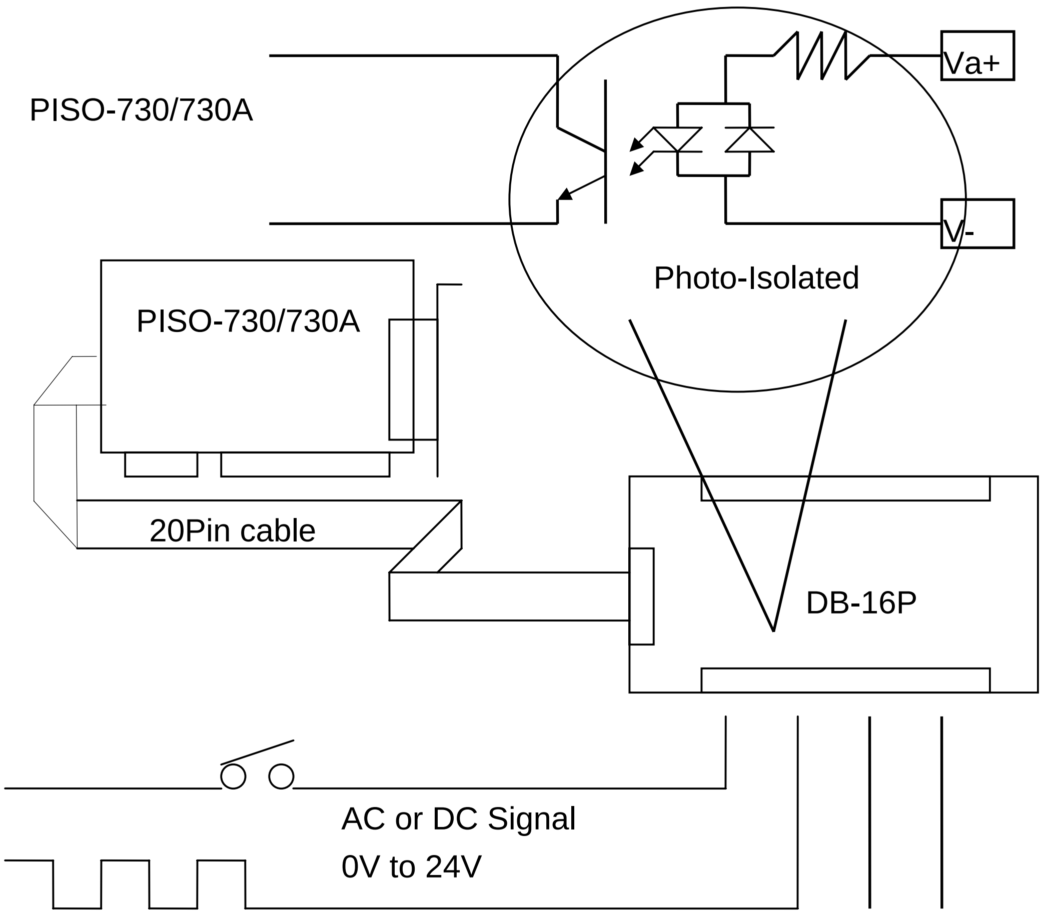
```
void interrupt irq_service(){
new_int_state=inportb(wBase+7)&0x03; /* read all interrupt state */
int_c=new_int_state^now_int_state;   /* compare which interrupt  */
                                     /* signal has changed       */

if ((int_c&0x1)!=0)              /* INT_CHAN_0 is active     */
  {
  if ((new_int_state&0x01)!=0)      /* now DI0 changes to high   */
    {
    CNT_H1++;
    }   else                     /* now DI0 changes to low    */
    {
    CNT_L1++;
    }   invert=invert^1;               /* to generate a high pulse */   }if ((int_c&0x2)!=0)   {   if
((new_int_state&0x02)!=0)      /* now DI1 change to high   */
    {
    CNT_H2++;
    }   else                     /* now DI1 changes to low    */
    {
    CNT_L2++;
    }   invert=invert^2;               /* to generate a high pulse */   }
now_int_state=new_int_state;
outportb(wBase+0x2a,invert);
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```

2.4 Daughter Boards

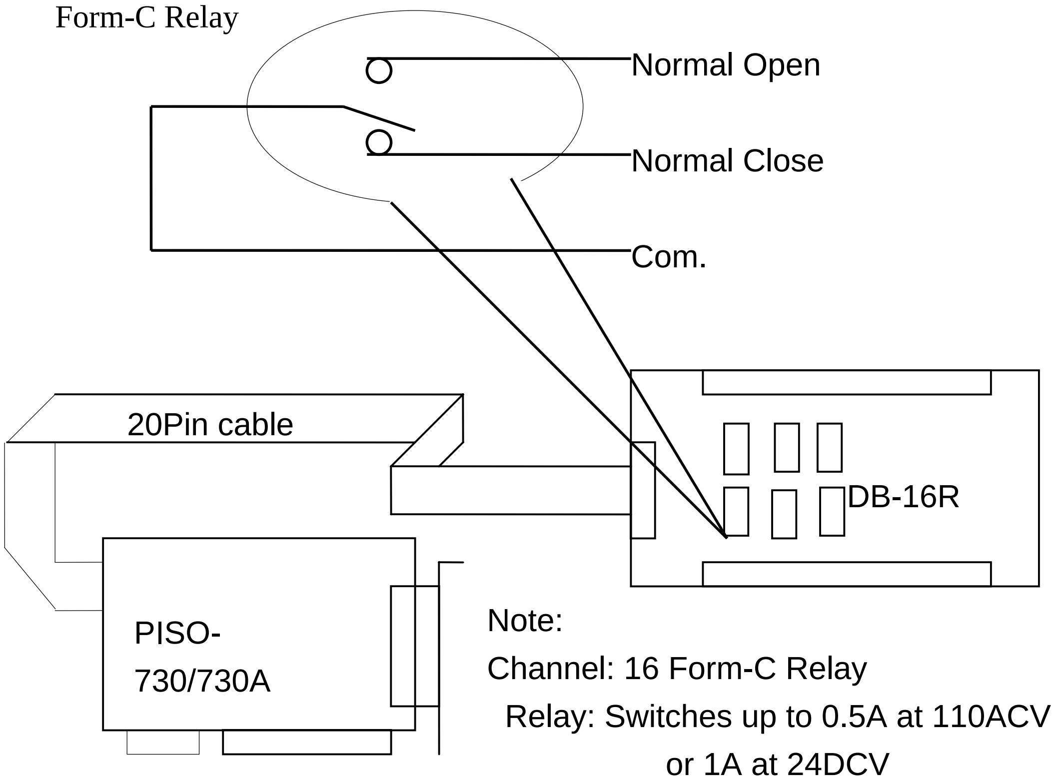
2.4.1 DB-16P Isolated Input Board

The DB-16P is a 16-channel isolated digital input daughter board. The optically isolated inputs of the DB-16P consist of a bi-directional photo-coupler with a resistor that acts as a current senser. Use the DB-16P to sense DC signals from TTL levels up to 24V or use the DB-16P to sense a wide range of AC signals. This board can also isolate the computer from large common-mode voltages, ground loops and transient voltage spikes that often occur in industrial environments.



2.4.2 DB-16R Relay Board

The DB-16R is a 16-channel relay output board, consisting of 16 form-C relays for optimum program-controlled, load switching efficiency. The relays are energized by applying a 12V/24V voltage signal to the appropriate relay channel on the 20-pin flat connector. There are 16 enunciator LEDs for each relay, which light when their associated relay is activated.

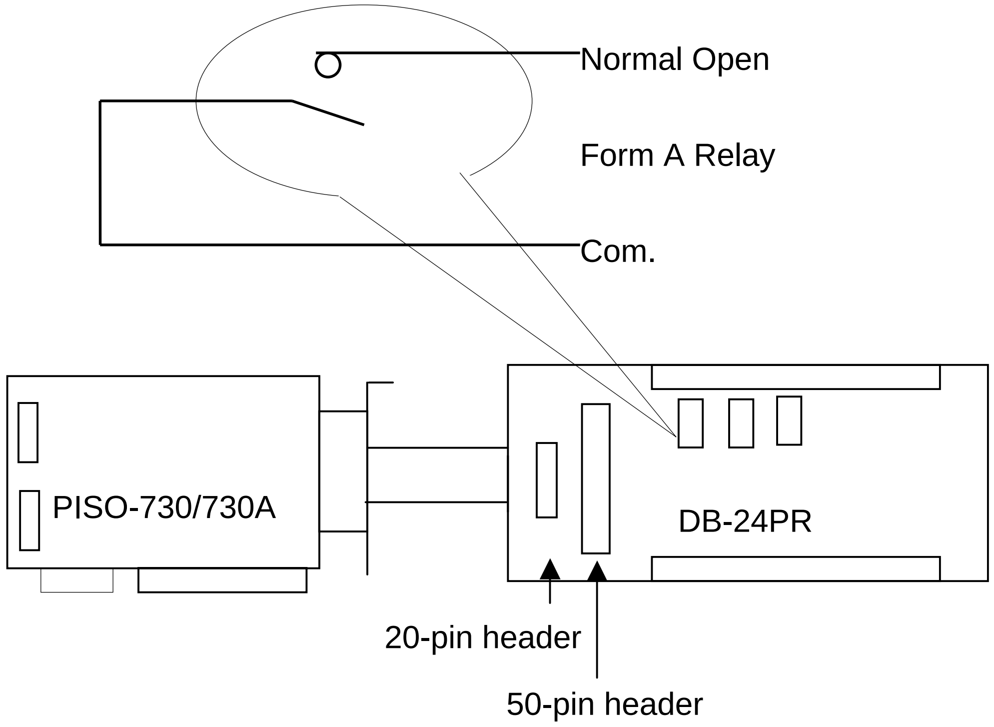


Note:
Channel: 16 Form-C Relay
Relay: Switches up to 0.5A at 110ACV or 1A at 24DCV

2.4.3 DB-24PR, DB-24POR, DB-24C

DB-24PR	24*power relay, 5A/250V
DB-24POR	24*photoMOS relay, 0.1A/350VAC
DB-24C	24*open collector, 100mA per channel, 30V max.

The DB-24PR, is a 24-channel power relay output board. It consists of 8 form-C and 16 form-A electromechanical relays providing efficient program-controlled load switching. Each relay contact can control a 5A load at 250ACV/30VDCV. The relay is energized by applying a 5-volt signal to the appropriate relay channel on the 20-pin flat cable connector (only 16 are used) or 50-pin flat cable connector (OPTO-22 compatible, for DIO-24 series). 24 enunciator LEDs, one for each relay, light when their associated relay is activated. To avoid overloading your power supply, this board needs a +12VDC or +24VDC external power supply.



Note:
50-Pin connector (OPTO-22 compatible), for DIO-24, DIO-48, DIO-144, PIO-D144, PIO-D96, PIO-D56, PIO-D48, PIO-D24
Channel: 16 Form-A Relays, 8 Form C Relays
Relay: Switches up to 5A at 110ACV / 5A at 30DCV

2.4.4 Daughter Board Comparison Table

	20-pin flat-cable header	50-pin flat-cable header	DB-37 header
DB-37	No	No	Yes
DN-37	No	No	Yes
ADP-37/PCI	No	Yes	Yes
ADP-50/PCI	No	Yes	No
DB-24P	No	Yes	No
DB-24PD	No	Yes	Yes
DB-16P8R	No	Yes	Yes
DB-24R	No	Yes	No
DB-24RD	No	Yes	Yes
DB-24C	Yes	Yes	Yes
DB-24PR	Yes	Yes	No
Db-24PRD	No	Yes	Yes
DB-24POR	Yes	Yes	Yes
DB-24SSR	No	Yes	Yes

NOTE: The PISO-730/730A has two 20-pin flat-cable headers, and one 37 pin D-type Connector.

2.5 Pin Assignment

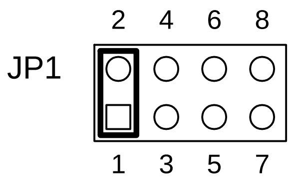
2.5.1 Isolated I/O connector

CON1: The 37 pins of the D-type female connector

Pin No.	Description	Pin No	Description
1	IDI0	20	IDI1
2	IDI2	21	IDI3
3	IDI4	22	IDI5
4	IDI6	23	IDI7
5	IDI8	24	IDI9
6	IDI10	25	IDI11
7	IDI12	26	IDI13
8	IDI14	27	IDI15
9	EI.COM1	28	EI.COM2
10	EO.COM1	29	IGND
11	IDO0	30	IDO1
12	IDO2	31	IDO3
13	IDO4	32	IDO5
14	IDO6	33	IDO7
15	IDO8	34	IDO9
16	IDO10	35	IDO11
17	IDO12	36	IDO13
18	IDO14	37	IDO15
19	EO.COM2		

2.5.2 JP1

<u>Note: Reserved</u>



2.5.3 TTL-level I/O connector

CON2: The 20 pins of the flat-cable connector

Pin no.	Description	Pin no.	Description
1	DI0	2	DI1
3	DI2	4	DI3
5	DI4	6	DI5
7	DI6	8	DI7
9	DI8	10	DI9
11	DI10	12	DI11
13	DI12	14	DI13
15	DI14	16	DI15
17	D.GND	18	D.GND
19	+5V	20	+12V

CON3: The 20 pins of the flat-cable connector

Pin no.	Description	Pin no.	Description
1	DO0	2	DO1
3	DO2	4	DO3
5	DO4	6	DO5
7	DO6	8	DO7
9	DO8	10	DO9
11	DO10	12	DO11
13	DO12	14	DO13
15	DO14	16	DO15
17	D.GND	18	D.GND
19	+5V	20	+12V

All signals are TTL compatible

3. I/O Control Register

3.1 How to Find the I/O Address

The plug & play BIOS will assign a proper I/O address to every PIO/PISO series card during the power-up stage. The fixed IDs of PIO/PISO series cards are given as follows:

PISO-730
<Rev1.0~Rev2.0>

- **Vender ID= 0xE159**
- **Device ID= 0x02**
- **Sub-Vendor ID= 0x80**
- **Sub-Device ID= 0x08**
- **Sub-Aux ID= 0x40**

PISO-730A
<Rev3.0>

- **Vender ID= 0xE159**
- **Device ID= 0x02**
- **Sub-Vendor ID= 0x80**
- **Sub-Device ID= 0x08**
- **Sub-Aux ID= 0x80**

PISO-730
<Rev3.0>

- **Vender ID= 0xE159**
- **Device ID= 0x01**
- **Sub-Vendor ID= 0xc2ff**
- **Sub-Device ID= 0x00**
- **Sub-Aux ID= 0x40**

PISO-730A
<Rev4.0>

- **Vender ID= 0xE159**
- **Device ID= 0x01**
- **Sub-Vendor ID= 0x62ff**
- **Sub-Device ID= 0x00**
- **Sub-Aux ID= 0x80**

We provide the following all necessary functions as follows:

1. **PIO_DriverInit(&wBoard, wSubVendor, wSubDevice, wSubAux)**
2. **PIO_GetConfigAddressSpace(wBoardNo,*wBase,*wIrq, *wSubVendor, *wSubDevice, *wSubAux, *wSlotBus, *wSlotDevice)**
3. **Show_PIO_PISO(wSubVendor, wSubDevice, wSubAux)**

All functions are defined in PIO.H. Refer to Chapter 4 for more information. Here important driver information:

1. Resource-allocated information:

- wBase : BASE address mapping in this PC
- wIrq: IRQ channel number allocated in this PC

2. PIO/PISO identification information:

- wSubVendor: subVendor ID of this board
- wSubDevice: subDevice ID of this board
- wSubAux: subAux ID of this board

3. PC physical slot information:

- wSlotBus: hardware slot ID1 in this PC slot position
- wSlotDevice: hardware slot ID2 in this PC slot position

The utility program, **PIO_PISO.EXE**, detects & shows all PIO/PISO cards installed in this PC. Refer to Sec. 4.1 for more information.

3.1.1 PIO_DriverInit

PIO_DriverInit(&wBoards, wSubVendor,wSubDevice,wSubAux)

- wBoards=0 to N → number of boards found in this PC
- wSubVendor → subVendor ID of board to find
- wSubDevice → subDevice ID of board to find
- wSubAux → subAux ID of board to find

This function detects all PIO/PISO series card in the system. Implemented is based on the PCI plug & play mechanism-1. It will find all installed PIO/PISO series cards in this system, and saves all their resource in the library.

Sample program 1: Finds all PISO-730/730A in this PC

```
wSubVendor=0x80; wSubDevice=8; wSubAux=0x40;  /* for PISO-730 */
                                   ; wSubAux=0x80;  /* for PISO-730A*/
wRetVal=PIO_DriverInit(&wBoards, wSubVendor,wSubDevice,wSubAux);
printf("There are %d PISO-730 Cards in this PC\n",wBoards);

/* step2: Save resource of all PISO-730/730A cards installed in this PC */
for (i=0; i<wBoards; i++)
  {   PIO_GetConfigAddressSpace(i,&wBase,&wIrq,&wID1,&wID2,&wID3,
                                   &wID4,&wID5);
  printf("\nCard_%d: wBase=%x, wIrq=%x", i,wBase,wIrq);
  wConfigSpace[i][0]=wBaseAddress;          /*save all resource of this card */
  wConfigSpace[i][1]=wIrq;                  /* save all resource of this card */
  }
```

Sample program 2: Find all PIO/PISO in this PC (Refer to Sec. 4.1 for more information)

```
wRetVal=PIO_DriverInit(&wBoards,0xff,0xff,0xff);  /*find all PIO_PISO*/
printf("\nThere are %d PIO_PISO Cards in this PC",wBoards);
if (wBoards==0 ) exit(0);

printf("\n-----------------------------------------------------");
for(i=0; i<wBoards; i++)
  {
  PIO_GetConfigAddressSpace(i,&wBase,&wIrq,&wSubVendor,
          &wSubDevice,&wSubAux,&wSlotBus,&wSlotDevice);

  printf("\nCard_%d:wBase=%x,wIrq=%x,subID=[%x,%x,%x],
          SlotID=[%x,%x]",i,wBase,wIrq,wSubVendor,wSubDevice,
          wSubAux,wSlotBus,wSlotDevice);
  printf(" --> ");
  ShowPioPiso(wSubVendor,wSubDevice,wSubAux);
  }
```

3.1.2 PIO_GetConfigAddressSpace

PIO_GetConfigAddressSpace(wBoardNo,*wBase,*wIrq, *wSubVendor, *wSubDevice, *wSubAux, *wSlotBus, *wSlotDevice)

- wBoardNo=0 to N → total number of N+1 boards found by PIO_DriveInit(….)
- wBase → base address of the board control word
- wIrq → allocates IRQ channel number of this board
- wSubVendor → subVendor ID of this board
- wSubDevice → subDevice ID of this board
- wSubAux → subAux ID of this board
- wSlotBus → hardware slot ID1 of this board
- wSlotDevice → hardware slot ID2 of this board

Use this function to save the resources of all PIO/PISO cards installed in this system. Afterward, the application program can directly control all functions of PIO/PISO series card.

Find the configure address space of PISO-730/730A:

```
/* step1: detect all PISO-730/730A cards first */
wSubVendor=0x80; wSubDevice=8; wSubAux=0x40;  /* for PISO-730 */
                                  ;wSubAux=0x80;  /* for PISO-730A */
wRetVal=PIO_DriverInit(&wBoards, wSubVendor,wSubDevice,wSubAux);
printf("There are %d PISO-730 Cards in this PC\n",wBoards);
/* step2: save resource of all PISO-730/730A cards installed in this PC */
for (i=0; i<wBoards; i++)
  {   PIO_GetConfigAddressSpace(i,&wBase,&wIrq,&t1,&t2,&t3,&t4,&t5);
   printf("\nCard_%d: wBase=%x, wIrq=%x", i,wBase,wIrq);
  wConfigSpace[i][0]=wBaseAddress;    /* save all resource of this card       */
  wConfigSpace[i][1]=wIrq;            /* save all resource of this card       */
  }
/* step3: control the PISO-730/730A directly */
wBase=wConfigSpace[0][0];          /* get base address the card_0       */
outport(wBase,1);                  /* enable all D/I/O operation of card_0 * /
wBase=wConfigSpace[1][0];          /* get base address the card_1       */
outport(wBase,1);                  /* enable all D/I/O operation of card_1       */
```

3.1.3 Show_PIO_PISO

Show_PIO_PISO(wSubVendor,wSubDevice,wSubAux)

- wSubVendor → subVendor ID of board to find
- wSubDevice → subDevice ID of board to find
- wSubAux → subAux ID of board to find

This function shows a text string for these special subIDs. This text string is the same as that defined in PIO.H

Here the demo program for this function:

```
wRetVal=PIO_DriverInit(&wBoards,0xff,0xff,0xff);  /*find all PIO_PISO*/
printf("\nThrer are %d PIO_PISO Cards in this PC",wBoards);
if (wBoards==0 ) exit(0);

printf("\n----------------------------------------------------");
for(i=0; i<wBoards; i++)
  {
  PIO_GetConfigAddressSpace(i,&wBase,&wIrq,&wSubVendor,
            &wSubDevice,&wSubAux,&wSlotBus,&wSlotDevice);

  printf("\nCard_%d:wBase=%x,wIrq=%x,subID=[%x,%x,%x],
            SlotID=[%x,%x]",i,wBase,wIrq,wSubVendor,wSubDevice,
            wSubAux,wSlotBus,wSlotDevice);
  printf(" --> ");
  ShowPioPiso(wSubVendor,wSubDevice,wSubAux);
  }
```

3.2 The Assignment of I/O Address

The plug & play BIOS assigns the proper I/O address to each PIO/PISO series card. If there is only one PIO/PISO board, the board is identified as card_0. If there are two PIO/PISO boards in the system, identifying card_0 becomes more difficult? The software driver can support a maximum of 16 boards. Therefore, you can install up to 16 boards of PIO/PSIO series in one PC system. Here how to identify each address.

The easiest way to identify which card is card_0 is to use wSlotBus & wSlotDevice as follows:

1. Remove all PISO-730/730A from this PC
2. Install one PISO-730/730A into the PC PCI_slot1, run PIO_PISO.EXE & record the wSlotBus1 & wSlotDevice1
3. Remove all PISO-730/730A from this PC
4. Install one PISO-730/730A into the PC PCI_slot2, run PIO_PISO.EXE & record the wSlotBus2 & wSlotDevice2
5. Repeat (3) & (4) for all PCI_slot? Record all wSlotBus? & wSlotDevice?

The records may be as follows:

PC PCI slot	WSlotBus	wSlotDevice
Slot_1	0	0x07
Slot_2	0	0x08
Slot_3	0	0x09
Slot_4	0	0x0A
PCI-BRIDGE		
Slot_5	1	0x0A
Slot_6	1	0x08
Slot_7	1	0x09
Slot_8	1	0x07

The above procedure will record all wSlotBus? & wSlotDevice? in this PC. These values will be mapped to this PC physical slot. This mapping will not be changed for any PIO/PISO cards. Use it to identify the specified PIO/PISO card as follows:

Step1: Record all wSlotBus? & wSlotDevice?

Step2: Use PIO_GetConfigAddressSpace(…) to get the specified card wSlotBus & wSlotDevice

Step3: Identify the specified PIO/PISO card by comparing the wSlotBus & wSlotDevice in step2 to step1.

3.3 The I/O Address Map

A PIO / PISO series card address is automatically assigned by the main board ROM BIOS. The I/O address can also be re-assigned if needed, I**t is strongly recommended not to change the I/O address. The plug&play BIOS assigns the proper I/O address for each PIO/PISO series card.** Here are the of PISO-730/730A I/O addresses:

Address	**Read**	**Write**
WBase+0	RESET\ control register	Same
WBase+2	Aux control register	Same
WBase+3	Aux data register	Same
WBase+5	INT mask control register	Same
WBase+7	Aux pin status register	Same
WBase+0x2a	INT polarity control register	Same
WBase+0xc0	IDI0~IDI7	IDO0~IDO7
WBase+0xc4	IDI8~IDI15	IDO8~IDO15
WBase+0xc8	DI0~DI7	DO0~DO7
Wbase+0xcc	DI8~DI15	DO8~DO15

Note. Refer to Sec. 3.1 for more information about wBase.

3.3.1 RESET\ Control Register

(Read/Write): wBase+0

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	RESET\

Note. Refer to Sec. 3.1 for more information about wBase.

When the PC is first powered-up, the RESET\ signal is in Low state. **This disables all D/I/O operations.** Please set the RESET\ signal to High state before issuing any D/I/O command.

```
outportb(wBase,1);          /* RESET\ = High → all D/I/O are enable now */
outportb(wBase,0);          /* RESET\ = Low  → all D/I/O are disable now */
```

3.3.2 AUX Control Register

(Read/Write): wBase+2

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Aux7	Aux6	Aux5	Aux4	Aux3	Aux2	Aux1	Aux0

Note. Refer to Sec. 3.1 for more information about wBase.

Aux?=0→ this Aux is used as a D/I

Aux?=1→ this Aux is used as a D/O

When the PC is first powered-on, all Aux? signal are in Low-state. all Aux? are designed as D/I for all PIO/PISO series. Please set all Aux? in D/I state.

3.3.3 AUX data Register

(Read/Write): wBase+3

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Aux7	Aux6	Aux5	Aux4	Aux3	Aux2	Aux1	Aux0

Note. Refer to Sec. 3.1 for more information about wBase.

When the Aux? is used as a D/O, this register controls the output state. This register has been designed for future extensions, so please don try to control this register now.

3.3.4 INT Mask Control Register

(Read/Write): wBase+5

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	EN1	EN0

Note. Refer to Sec. 3.1 for more information about wBase.

EN0/1=0→ Disables INT_CHAN_0/1 as a interrupt signal (default)
EN0/1=1→ Enables INT_CHAN_0/1 as a interrupt signal

```
outportb(wBase+5,0);     /* Disables all interrupts                 */
outportb(wBase+5,1);     /* Enables interrupt of INT_CHAN_0         */
outportb(wBase+5,2);     /* Enables interrupt of INT_CHAN_1         */
outportb(wBase+5,3);     /* Enables all two channels of interrupt   */
```

Refer to the following demo program for more information:
DEMO3.C → For INT_CHAN_0 only (initial high state)
DEMO4.C → For INT_CHAN_0 only (initial low state)
DEMO5.C → For multi-channel interrupt sources

3.3.5 Aux Status Register

(Read/Write): wBase+7

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Aux7	Aux6	Aux5	Aux4	Aux3	Aux2	Aux1	Aux0

Note. Refer to Sec. 3.1 for more information about wBase.

Aux0=INT_CHAN_0, Aux1=INT_CHAN_1, Aux7~4=Aux-ID. Refer to Sec. 4.1 for more information. The Aux0~1 are used as interrupt sources. The interrupt service routine has to read this register for interrupt source identification. Refer to Sec. 2.5 for more information.

3.3.6 Interrupt Polarity Control Register

(Read/Write): wBase+0x2A

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	INV1	INV0

Note. Refer to Sec. 3.1 for more information about wBase.

INV0/1=0→ Selects the inverted signal from INT_CHAN_0/1

INV0/1=1→ Selects the non-inverted signal from INT_CHAN_0/1

```
outportb(wBase+0x2a,0);   /* Selects the invert input from all 2 channels       */
outportb(wBase+0x2a,0x3);/* Selects the non-invert input from all 2 channels */

outportb(wBase+0x2a,0x2);/* Selects the inverted input of INT_CHAN_0         */
                          /* Selects the non-inverted input of INT_CHAN_1  */
```

Refer to Sec. 2.6.7 for more information.
Refer to DEMO5.C for more information.

3.3.7 I/O Data Register

(Read/Write): wBase+0xC0

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IDI7	IDI6	IDI5	IDI4	IDI3	IDI2	IDI1	IDI0

(Read/Write): wBase+0xC4

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IDI15	IDI14	IDI13	IDI12	IDI11	IDI10	IDI9	IDI8

(Read/Write): wBase+0xC8

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0

(Read/Write): wBase+0xCC

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI15	DI14	DI13	DI12	DI11	DI10	DI9	DI8

Note. Refer to Sec. 3.1 for more information about wBase.

```
outportb(wBase+0xc0,0xff);          /* Writes 0xff to IDO0~IDO7       */
DiValue=inportb(wBase+0xc0);        /* Reads states from IDI0~IDI7    */

outportb(wBase+0xc8,0x55);          /* Writes 0x55 to DO0~DO7         */
DiValue=inportb(wBase+0xcc);        /* Reads states from DI8~DI15     */
```

4. Demo Program

It is recommended to read the release note first. All of the following important information will be given in release note:

1. The location of the software driver & utility
2. **Installing the software & utility**
3. The location of the diagnostic program
4. FAQ

There are many demo programs given in the company floppy disk or CD. After the software installation, the driver will be installed as follows:

- \TC*.* → for Turbo C 2.xx or above
- \MSC*.* → for MSC 5.xx or above
- \BC*.* → for BC 3.xx or above

- \TC\LIB*.* → for TC library
- \TC\DEMO*.* → for TC demo program
- \TC\DIAG*.* → for TC diagnostic program

- \TC\LIB\Large*.* → TC large model library
- \TC\LIB\Huge*.* → TC huge model library
- \TC\LIB\Large\PIO.H → TC declaration file
- \TC\\LIB\Large\TCPIO_L.LIB → TC large model library file
- \TC\LIB\Huge\PIO.H → TC declaration file
- \TC\\LIB\Huge\TCPIO_H.LIB → TC huge model library file

- \MSC\LIB\Large\PIO.H → MSC declaration file
- \MSC\LIB\Large\MSCPIO_L.LIB → MSC large model library file
- \MSC\LIB\Huge\PIO.H → MSC declaration file
- \MSC\\LIB\Huge\MSCPIO_H.LIB → MSC huge model library file

- \BC\LIB\Large\PIO.H → BC declaration file
- \BC\LIB\Large\BCPIO_L.LIB → BC large model library file
- \BC\LIB\Huge\PIO.H → BC declaration file
- \BC\\LIB\Huge\BCPIO_H.LIB → BC huge model library file

NOTE: The library is available for all PIO/PISO series cards.

4.1 PIO_PISO

```
/* -----------------------------------------------------------------------*/
/* Find all PIO_PISO series cards in this PC system       */
/* step 1 : plug all PIO_PISO cards into PC               */
/* step 2 : run PIO_PISO.EXE                              */
/* ----------------------------------------------------------------------*/

#include "PIO.H"

WORD wBase,wIrq;
WORD wBase2,wIrq2;

int main()
{
int  i,j,j1,j2,j3,j4,k,jj,dd,j11,j22,j33,j44;
WORD wBoards,wRetVal;
WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
char c;
float ok,err;

clrscr();
wRetVal=PIO_DriverInit(&wBoards,0xff,0xff,0xff);  /*for PIO-PISO*/
printf("\nThere are %d PIO_PISO Cards in this PC",wBoards);
if (wBoards==0 ) exit(0);

printf("\n-----------------------------------------------------");
for(i=0; i<wBoards; i++)
  {
  PIO_GetConfigAddressSpace(i,&wBase,&wIrq,&wSubVendor,
           &wSubDevice,&wSubAux,&wSlotBus,&wSlotDevice);

  printf("\nCard_%d:wBase=%x,wIrq=%x,subID=[%x,%x,%x],
           SlotID=[%x,%x]",i,wBase,wIrq,wSubVendor,wSubDevice,
           wSubAux,wSlotBus,wSlotDevice);
  printf(" --> ");
  ShowPioPiso(wSubVendor,wSubDevice,wSubAux);
  }

PIO_DriverClose();
}
```

NOTE**: the PIO_PISO.EXE is valid for all PIO/PISO cards**. It is located in \TC\DIAG\ directory. Execute the PIO_PISO.EXE to get the following information:

- A list all PIO/PISO cards installed in this PC
- A list all resources allocated to every PIO/PISO cards
- A list the wSlotBus & wSlotDevice for specified PIO/PISO card identification. (Refer to Sec. 3.2 for more information)

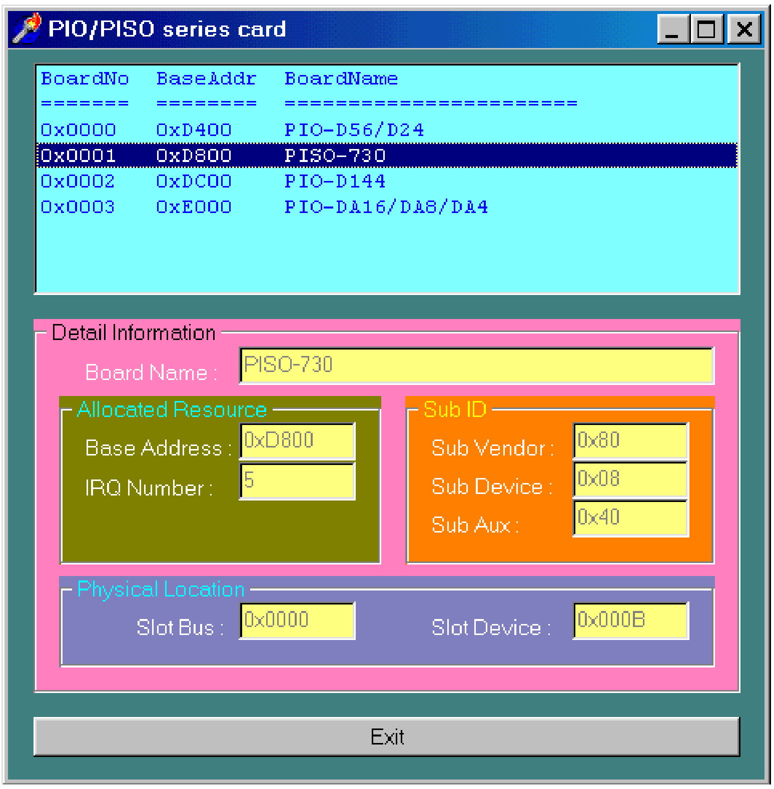
4.1.1 PIO_PISO.EXE for Windows

PIO_PISO.EXE is a software utility for Windows95/98. For detailed information about this file, please refer to the readme.txt" of your Windows 95/98 development toolkit. It is useful for all PIO/PISO series cards.

Here are the setup steps from the CD-ROM:

- Step1: Toolkit(Software)/Manuals
- Step2: I Agree
- Step3: PCI Bus DAQ Card
- Step4: PIO_PISO
- Step5: Installs Toolkits for Windows95/98
- Step6: After installation, this program will be extracted in user define directory.

After executing the utility, all detailed information for all PIO/PISO cards installed in the PC will be shown as follows:



4.2 DEMO1

```
/* DEMO1.C : PISO-730/730A D/O demo            */
/* step 1  : connect CON3 to DB-16R             */
/* step 2  : run DEMO1.EXE                      */
/* ------------------------------------------------------------ */
#include "PIO.H"
void piso_730_do(long lDoValue);
void piso_730_ido(long lDoValue);
WORD wBase,wIrq;
int main()
{
int  i,j,k1,k2,l1,l2,jj,dd,j1,i1,j2,i2;
WORD wBoards,wRetVal,t1,t2,t3,t4,t5;
WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
long lOutPad1,lOutPad2;
char c;

clrscr();
/* step 1: find address-mapping of PIO/PISO cards              */
wRetVal=PIO_DriverInit(&wBoards,0x80,0x08,0x40);  /* for PISO-730 */
                                   ,0x80);  /* for PISO-730A*/
printf("\nThere are %d PISO-730/730A Cards in this PC",wBoards);
if (wBoards==0) exit(0);
printf("\n--------------- The Configuration Space ---------------");
for(i=0; i<wBoards; i++)
   {
   PIO_GetConfigAddressSpace(i,&wBase,&wIrq,&wSubVendor,&wSubDevice,
   &wSubAux,&wSlotBus,&wSlotDevice);
   printf("\nCard_%d: wBase=%x,wIrq=%x,subID=[%x,%x,%x],SlotID=
         [%x,%x]",i,wBase,wIrq,wSubVendor,wSubDevice,wSubAux,
         wSlotBus,wSlotDevice);
   printf(" --> ");
   ShowPioPiso(wSubVendor,wSubDevice,wSubAux);
   }

PIO_GetConfigAddressSpace(0,&wBase,&wIrq,&t1,&t2,&t3,&t4,&t5);
/* step 2: enable all D/I/O port                              */
outportb(wBase,1);                          /* enable D/I/O */

printf("\n\n");
lOutPad1=1;
lOutPad2=0x8000;
for(;;)
   {
   gotoxy(1,6);
   piso_730_do(lOutPad1);
   printf("\nOutput DO[0..15] = [%4lx]",lOutPad1);
   piso_730_ido(lOutPad2);
   printf("\nOutput IDO[0..15] = [%4lx]",lOutPad2);

   delay(12000);
   lOutPad1=((lOutPad1<<1)&0xffff);
   lOutPad2=((lOutPad2>>1)&0xffff);

   if (lOutPad1==0) {lOutPad1=1;lOutPad2=0x8000;}
   if (kbhit()!=0) break;
   }
PIO_DriverClose();
```

```
}

/* -------------------------------------------------------------- */
void piso_730_do(long lDoValue)
{
outportb(wBase+0xc8,(lDoValue&0xff));
outportb(wBase+0xcc,((lDoValue>>8)&0xff));
}
/* -------------------------------------------------------------- */
void piso_730_ido(long lDoValue)
{
outportb(wBase+0xc0,(lDoValue&0xff));
outportb(wBase+0xc4,((lDoValue>>8)&0xff));
}
```

4.3 DEMO2

```
/* DEMO2.C : PISO-730/730A D/I/O demo          */
/* step 1  : connect  DO[0..15] to  DI[0..15],         */
/*                  IDO[0..15] to IDI[0..15]                  */
/* step 2  : run DEMO2.EXE                              */
/* ------------------------------------------------------------- */
#include "PIO.H"
long piso_730_di(void);
long piso_730_idi(void);
WORD wBase,wIrq;
int main()
{
int  i,j,k,k1,k2,l1,l2,jj,dd,j1,i1,j2,i2;
WORD wBoards,wRetVal,t1,t2,t3,t4,t5;
WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
long lOutPad1,lOutPad2,lInPad1,lInPad2;
char c;
clrscr();
/* step 1: find address-mapping of PIO/PISO cards                 */
          .
          .
/* step 2: enable all D/I/O port                              */
outportb(wBase,1);                            /* enable D/I/O */
lOutPad1=0x0001;
lOutPad2=0x8000;
for(;;)
   {gotoxy(1,8);
   piso_730_do(lOutPad1);
   lInPad1=piso_730_di();
   piso_730_ido(lOutPad2);
   delay(10000);
   lInPad2=piso_730_idi();
   printf("\n DO[0..15]=[%4lx] ,   DI[0..15]=[%4lx]",lOutPad1,lInPad1);
   printf("\nIDO=[%4lx],!IDI=[%4lx]",lOutPad2,(~lInPad2&0xffff));
   lOutPad1=(lOutPad1<<1)&0xffff;
   lOutPad2=(lOutPad2>>1)&0xffff;
   if (lOutPad1==0) lOutPad1=1;
   if (lOutPad2==0) lOutPad2=0x8000;
   if (kbhit()!=0) break;
   }
PIO_DriverClose();
}
/* -------------------------------------------------------------- */
long piso_730_di(void)
{
long lDiValue;
lDiValue=(inportb(wBase+0xcc)<<8);
lDiValue=(lDiValue|(inportb(wBase+0xc8)))&0xffff;
return(lDiValue);
}
/* -------------------------------------------------------------- */
long piso_730_idi(void)
{
long lDiValue;
lDiValue=(inportb(wBase+0xc4)<<8);
lDiValue=(lDiValue|(inportb(wBase+0xc0)))&0xffff;
return(lDiValue);
}
```

4.4 DEMO3

```
/* DEMO3.C : PISO-730/730A interrupt (DI0 initial high)    */
/* step 1  : DI0 to function generator                      */
/* step 2  : run DEMO3.EXE                                  */
/* -------------------------------------------------------------------- */
#include "PIO.H"
#define A1_8259 0x20
#define A2_8259 0xA0
#define EOI     0x20

WORD init_high();
void interrupt (*oldfunc) ();
static void interrupt irq_service();
int COUNT_L,COUNT_H,irqmask,now_int_state;

void piso_730_do(long lDoValue);
long piso_730_di(void);
void piso_730_ido(long lDoValue);
long piso_730_idi(void);

WORD wBase,wIrq;

int main()
{
int  i,j,k,k1,k2,l1,l2,jj,dd,j1,i1,j2,i2;
WORD wBoards,wRetVal,t1,t2,t3,t4,t5;
WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
char c;

clrscr();
/* step 1: find address-mapping of PIO/PISO cards              */
        .
        .
/* step 2: enable all D/I/O port                         */
outportb(wBase,1);                          /* enable D/I/O */

init_high();

printf("\n\n***** show the count of Low_pulse *****\n");
for(;;)
  {
  gotoxy(1,8);
  printf("\nCOUNT_L=[%5d]",COUNT_L);
  if (kbhit()!=0) break;
  }

disable();

outportb(wBase+5,0);                    /* disable all interrupt */
if (wIrq<8)
  {
  setvect(wIrq+8,oldfunc);
  }
else
  {
  setvect(wIrq-8+0x70,oldfunc);
  }
PIO_DriverClose();
```

```
}
/* -------------------------------------------------------------- */
WORD init_high()
{
DWORD dwVal;

disable();
outportb(wBase+5,0);                    /* disable all interrupt */

if (wIrq<8)
   {
   oldfunc=getvect(wIrq+8);
   irqmask=inportb(A1_8259+1);
   outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
   setvect(wIrq+8, irq_service);
   }
else
   {
   oldfunc=getvect(wIrq-8+0x70);
   irqmask=inportb(A1_8259+1);
   outportb(A1_8259+1,irqmask & 0xfb);                  /* IRQ2 */
   irqmask=inportb(A2_8259+1);
   outportb(A2_8259+1,irqmask & (0xff ^ (1 << (wIrq-8))));
   setvect(wIrq-8+0x70, irq_service);
   }

outportb(wBase+0x2a,0);               /* invert DI0             */

now_int_state=0x1;                  /* now DI0 is high        */
outportb(wBase+5,0x1);                /* enable DI0 interrupt    */
enable();
}

/* -------------------------------------------------------------- */
void interrupt irq_service()
{
if (now_int_state==1)            /* now DI0 change to low         */
   {                       /* INT_CHAN_0 = !DI0            */
   COUNT_L++;                   /* find a low pulse (DI0)        */
   if ((inportb(wBase+7)&1)==0)  /* DI0 still fixed in low        */
      {                       /* need to generate a high pulse */
      outportb(wBase+0x2a,1);    /* INV0 select noninverted input */
      now_int_state=0;           /* now DI0=low                  */
      }
   else now_int_state=1;         /* now DI0=High                 */
   }
else                          /* now DI0 change to high        */
   {                       /* INT_CHAN_0 = DI0              */
   COUNT_H++;                   /* find a high pulse (DI0)       */
   if ((inportb(wBase+7)&1)==1)  /* DI0 still fixed in high       */
      {                       /* need to generate a high pulse */
      outportb(wBase+0x2a,0);    /* INV0 select inverted input    */
      now_int_state=1;           /* now DI0=high                 */
      }
   else now_int_state=0;         /* now DI0=low                  */
   }
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```

4.5 DEMO4

```
/* DEMO4.C : PISO-730/730A Interrupt (DI0 initial low)      */
/* step 1  : DI0 to function generator                      */
/* step 2  : run DEMO4.EXE                                  */
/* -------------------------------------------------------------------   */
#include "PIO.H"

#define A1_8259 0x20
#define A2_8259 0xA0
#define EOI     0x20

WORD init_low();
void interrupt (*oldfunc) ();
static void interrupt irq_service();
int COUNT_L,COUNT_H,irqmask,now_int_state;

void piso_730_do(long lDoValue);
long piso_730_di(void);
void piso_730_ido(long lDoValue);
long piso_730_idi(void);

WORD wBase,wIrq;

int main()
{
int  i,j,k,k1,k2,l1,l2,jj,dd,j1,i1,j2,i2;
WORD wBoards,wRetVal,t1,t2,t3,t4,t5;
WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
char c;


clrscr();
/* step 1: find address-mapping of PIO/PISO cards                */
       .
       .
/* step 2: enable all D/I/O port                          */
outportb(wBase,1);                            /* enable D/I/O */

init_Low();

printf("\n\n***** show the count of High_pulse *****\n");
for(;;)
  {
  gotoxy(1,8);
  printf("\nCOUNT_H=[%5d]",COUNT_H);
  if (kbhit()!=0) break;
  }
disable();
outportb(wBase+5,0);                    /* disable all interrupt */
if (wIrq<8)
  {
  setvect(wIrq+8,oldfunc);
  }
else
  {
  setvect(wIrq-8+0x70,oldfunc);
  }
PIO_DriverClose();
```

```
}
/* -------------------------------------------------------------- */
WORD init_low()
{
DWORD dwVal;

disable();
outportb(wBase+5,0);                    /* disable all interrupt */

if (wIrq<8)
   {
   oldfunc=getvect(wIrq+8);
   irqmask=inportb(A1_8259+1);
   outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
   setvect(wIrq+8, irq_service);
   }
else
   {
   oldfunc=getvect(wIrq-8+0x70);
   irqmask=inportb(A1_8259+1);
   outportb(A1_8259+1,irqmask & 0xfb);                   /* IRQ2 */
   irqmask=inportb(A2_8259+1);
   outportb(A2_8259+1,irqmask & (0xff ^ (1 << (wIrq-8))));
   setvect(wIrq-8+0x70, irq_service);
   }
outportb(wBase+0x2a,1);                /* non-invert DI0          */
now_int_state=0x0;                     /* now DI0 is low          */
outportb(wBase+5,0x1);                 /* enable DI0 interrupt    */
enable();
}
/* -------------------------------------------------------------- */
void interrupt irq_service()
{
if (now_int_state==1)            /* now DI0 change to low         */
   {                        /* INT_CHAN_0 = !DI0             */
   COUNT_L++;                   /* find a low pulse (DI0)        */
   if ((inportb(wBase+7)&1)==0)  /* DI0 still fixed in low        */
      {                     /* need to generate a high pulse */
      outportb(wBase+0x2a,1);    /* INV0 select noninverted input */
      now_int_state=0;          /* now DI0=low                   */
      }
   else now_int_state=1;         /* now DI0=High                  */
   }
else                          /* now DI0 change to high        */
   {                        /* INT_CHAN_0 = DI0              */
   COUNT_H++;                   /* find a high pulse (DI0)       */
   if ((inportb(wBase+7)&1)==1)  /* DI0 still fixed in high       */
      {                     /* need to generate a high pulse */
      outportb(wBase+0x2a,0);    /* INV0 select inverted input    */
      now_int_state=1;          /* now DI0=high                  */
      }
   else now_int_state=0;         /* now DI0=low                   */
   }
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```

4.6 DEMO5

```
/* DEMO5.C : PISO-730/730A Interrupt (Multi interrupt source)     */
/*           DI0 : initial low ,  DI1 : initial high               */
/* step 1  : connect DI0 & DI1 to function generator              */
/* step 2  : run DEMO5.EXE                                        */
/* ---------------------------------------------------------------------------      */
#include "PIO.H"

#define A1_8259 0x20
#define A2_8259 0xA0
#define EOI     0x20

WORD init();
void interrupt (*oldfunc) ();
static void interrupt irq_service();
int  irqmask,now_int_state,new_int_state,invert,int_c,int_num;
int  CNT_L1,CNT_L2,CNT_H1,CNT_H2;

WORD wBase,wIrq;

int main()
{
int  i,j,k;
WORD wBoards,wRetVal,t1,t2,t3,t4,t5;
WORD wSubVendor,wSubDevice,wSubAux,wSlotBus,wSlotDevice;
char c;

clrscr();
/* step 1: find address-mapping of PIO/PISO cards                */
          .
          .
/* step 2: enable all D/I/O port                              */
outportb(wBase,1);                              /* enable D/I/O */
init();
printf("\n\n***** show the count of High_pulse *****\n");
for(;;)
   {
   gotoxy(1,8);
   printf("\nCNT_L1,CNT_L2=[%5d,%5d]",CNT_L1,CNT_L2);
   printf("\nCNT_H1,CNT_H2=[%5d,%5d]",CNT_H1,CNT_H2);
   if (kbhit()!=0) break;
   }
disable();
outportb(wBase+5,0);                    /* disable all interrupt */
if (wIrq<8)
   {
   setvect(wIrq+8,oldfunc);
   }
else
   {
   setvect(wIrq-8+0x70,oldfunc);
   }
PIO_DriverClose();
}
/* ----------------------------------------------------------------- */
WORD init()
{
DWORD dwVal;
```

```
disable();
outportb(wBase+5,0);                    /* disable all interrupt */
if (wIrq<8)
  {
  oldfunc=getvect(wIrq+8);
  irqmask=inportb(A1_8259+1);
  outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
  setvect(wIrq+8, irq_service);
  }
else
  {
  oldfunc=getvect(wIrq-8+0x70);
  irqmask=inportb(A1_8259+1);
  outportb(A1_8259+1,irqmask & 0xfb);                  /* IRQ2 */
  irqmask=inportb(A2_8259+1);
  outportb(A2_8259+1,irqmask & (0xff ^ (1 << (wIrq-8))));
  setvect(wIrq-8+0x70, irq_service);
  }
invert=0x1;
outportb(wBase+0x2a,invert);          /* non-invert  DI0         */
                                   /*     invert  DI1         */
now_int_state=0x2;                  /* now  DI0 is low        */
                                   /* now  DI1 is high       */
outportb(wBase+5,0x3);                /* enable all interrupt    */
enable();
}
/* ------------------------------------------------------------- */
void interrupt irq_service()
{
int_num++;
new_int_state=inportb(wBase+7)&0x3;
int_c=new_int_state^now_int_state;
if ((int_c&0x1)!=0)             /* now INT_CHAN_0 change to high */
  {
  if ((new_int_state&0x01)!=0)
    {
    CNT_H1++;
    }
  else                        /* now INT_CHAN_0 change to low  */
    {    CNT_L1++;    }
  invert=invert^1;             /* generate a high pulse        */
  }
if ((int_c&0x2)!=0)             /* now INT_CHAN_1 change to high */
  {
  if ((new_int_state&0x02)!=0)
    {     CNT_H2++;     }
  else                        /* now INT_CHAN_1 change to low  */
    {     CNT_L2++;     }
  invert=invert^2;             /* generate a high pulse        */
  }
now_int_state=new_int_state;
outportb(wBase+0x2a,invert);
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```